PHP Vulnerabilities in Web Servers

Written By:
David K. Liefer
Steven K. Ziegler

Abstract:

The Internet has grown to be hugely popular and used by people of all different backgrounds and professions. Individual web pages are created by just about everyone whether or not they have any development experience or not. PHP (PHP: Hypertext Preprocessor) is one of the more popular scripting languages used by beginners and advanced users. It is an attractive alternative to Java for the novice user but little do they know there are some frightening vulnerabilities that can be exploited by clients looking to cause problems or gain access to private information or resources that cannot be tied to them. A few of these exploits include remote and local file inclusion or execution. Through these basics types of vulnerabilities a malicious client could gain complete access to a web server. To avoid these attacks a web developer needs to take care when writing PHP scripts. The most common mistake made by developers is to unknowingly expose internal variables to clients or when access is needed not properly sanitizing them to ensure the values make sense for the context in which they are being used. If un-sanitized variables are used in conjunction with certain PHP function calls private files can be accessed or remote files can be uploaded and executed. A few of the more dangerous PHP functions calls are _GET[] or passthru(). These functions need to be used with care or disabled by the system administrator to avoid problems with badly written PHP scripts. To aid in the detection of potential vulnerabilities the authors of this paper implemented a Vulnerability Detection Tool (VDT). The tool is a java based program which takes a set of user defined rules then uses these rules to parse through the contents of every file in a web development directory. If a defined rule is violated a report giving the filename, line number, and severity is displayed in the progress window of the GUI. Readers are encouraged to evaluate the tool, provide feedback to the authors, and when so motivated submit improvements to the tool.

Table of Contents:

---

1. Introduction

In the early days of the Internet most web development was done by professionals. Since then an array of web development tools, several scripting languages, and easily configurable web server software has made it easier for the novice to create and host their own website's. One of the more popular scripting languages is PHP. It has a significant user base and thousands of free scripts can be found all over the internet to perform all kinds of useful functions. It is an attractive alternative to Java for the novice user but little do they know there are some frightening vulnerabilities that can exploited by clients looking to cause problems or gain access to private information or resources that cannot be tied to them. In this paper, the basics behind the PHP scripting language and Apache web server architecture will be outlined. The latter mainly to understand how requests and data get forwarded from the web server to the underlining PHP module for interpretation and then passed back to the web server core where it is sent to the client requesting the information. Any web server could have been used for this paper, but the Apache web server was chosen dues to its popularity and availability as an open source product. Next, an investigation of the various PHP vulnerabilities will be conducted to provide readers with information that will help them write PHP scripts that are not easily exploited. To help find these vulnerabilities in website source code the writers of this paper have created a simple yet flexible tool that will recursively check selected directories for files with certain extensions to see if they have violated any of the predefined or user defined rules. When a rule is violated the severity is reported along with the filename and line number. The tool is aptly named the Vulnerability Detection Tool and its design and use is also outlined in this paper.

---

2. Background

The subject of this paper is to find PHP vulnerabilities in web servers. The web server we chose to use for this project is Apache, which is an open source product produced by the Apache Software Foundation. A little background information on PHP and the Apache Web server is probably warranted.

2.1 PHP

The roots of PHP are quite simple and originate with one man. His name was Rasmus Lerdorf and in 1995 he wrote a simple set of Perl scripts to track accesses of his online resume. He named these scripts "Personal Home Page Tools". Over time the size and number of Perl scripts got rather large and it was clear that an implementation in a standard programming language would be required to make it more scalable and easier to maintain. He chose the C programming language and made the application and source available to everyone.

He called the application "Personal Home Page / Forms Interpreter" or PHP/FI. The new implementation gave users the ability to communicate with databases and make simple dynamic web applications. Over the years, PHP/FI became quite popular and in 1997 the second version of PHP/FI was released. This version incorporated fixes and enhancements from the user community. The official release date for PHP/FI 2.0 was November 1997 but its life would be short lived because PHP/FI would receive a major overhaul and name change from some new developers.

A couple of students attempted to use PHP/FI for a university project but realized PHP/FI was not powerful enough to support the eCommerce application they developed for their project. Their names were Andi Gutmans and Zeev Suraski. Instead of abandoning PHP/FI they decided to redesign it so it fulfilled the needs for their project. The new version 3.0 was released, with the cooperation of Rasmus Lerdorf, as a successor to PHP/FI and was renamed to simply PHP. The new acronym is meant to be recursive for PHP: Hypertext Preprocessor. The new version contained many enhancements but one of its strongest was extensibility. This feature alone attracted many developers to create extension modules that added to the functionality of PHP and also its popularity. The new release also provided a solid infrastructure for different databases, protocols, and APIs. In addition, its object oriented support and much more consistent and verbose language syntax made it a powerful web development application.

Once renamed and released Andi Gutmans and Zeev Suraski took over development of PHP. The new version 3.0 added a lot of new functionality but did not do it very efficiently nor was the architecture as modular as the authors would have liked. To solve this problem a re-write of the PHP core was required.

The re-write of the PHP core was completed and ready to release as version 4 in May of 2000. This version contained what the authors dubbed as the "Zend Engine" and was named using parts of both their first names. The "Zend Engine" met all the design requirements for performance and modularity. In addition, it added many new features such as support of more Web servers, HyperText Transfer Protocol (HTTP) sessions, output buffering, and fixes for security vulnerabilities dealing with handling user input. The next major version was released in July 2004 and contains the second version of the Zend Engine, more fixes, and additional functionality. The latest released version, as of this writing is 5.2.5. Now that we know about the history of PHP lets look at the language itself to better understand its popularity. In section 3 of this paper some of the vulnerabilities of PHP will be investigated.

The main idea behind PHP is to be able to write PHP scripts embedded within Hyper Text Markup Language (HTML). An example of a simple script to echo something onto the web browser is shown in Figure 1 below.

```
<html>
    <head>
        <title>Example</title>
    </head>
    <body>
        <?php
            Echo “Hello World”;
        ?>
    </body>
</html>
```

Figure 1: Simple PHP Example

In this simple example the PHP tags allow the web server to know when to send the code to the PHP module running in the background. When the PHP start tag is encountered the web server application that is

interpreting the HTML code knows to pass this to the PHP module. The PHP module then takes the code and creates HTML code to write out "Hello World" to the client's browser screen. It is very powerful and different from client side java scripts because the PHP code is executed on the server and not the client. There are three main areas of Web development that PHP can handle. These are server-side scripting, command line scripting, and writing desktop applications. The last type in the list, desktop applications, is not an area that PHP excels as pointed out by the creators on the PHP website. However, there is an extension for creating Graphical User Interfaces (GUI) called PHP-GTK [Oglio et al. 2006]. Additionally, PHP can be used by the most popular web servers on all of the main stream operating systems. To understand some of the vulnerabilities examined in later sections of this paper we will need to dig a little deeper into basic PHP syntax.

The first time you look at PHP code you will notice some similarities to other scripting languages like Perl or Tcl. Figure 2 shows how variables are declared in PHP scripts.

```
$strVar = "I am a string";
$intVar = 12;
$floatVar = 12.0;
```

Figure 2: Variable Declaration in PHP

The type of variable is detected by the way it is assigned a value and could change over time as the variable is used in different contexts. On the surface this seems pretty powerful but it could get you into trouble if care is not taken when using the variable. Also, notice that PHP is like most programming languages in the way a line ends with a semi-colon. To declare an array it takes a little more work and the variable is fixed to be an array for life as shown in Figure 3.

```
$array1D = array("hello" => "again", "six" => 6);
$array2D = array("col" => array( "row1" => 123));
```

Figure 3: Array Declaration in PHP

An array in PHP is actually implemented as an ordered map similar to a map in the Standard Template Library (STL) for programming languages like C/C++ and Java. This means they will resize themselves automatically and everything can be indexed using a key value. The difference between this and an STL map is the key is not a fixed type. It can be any of the allowable PHP types and can differ from element to element in the same map. The same can be said of the value stored at that keyed location. This would appear to be a very powerful feature. The next thing to examine is how to operate on the variables using expressions and control structures.

The way variables can be operated on is very similar to most programming languages. Given the similarities a listing of these operators is not necessary but before writing any PHP scripts it is suggested to review them via online documentation [Achour, et al. 2007]. The same can be said of the control structures with exception of a few new types that are unique to the PHP language. Table 1 contains the list of unique control structures and a short description of their operation. A few of these typically are not considered control structures in most programming languages but because this is a scripting language they are defined in this manner. Now that we know a little about the different types of constructs available in PHP lets examine how functions are created within PHP.

Table 1: Unique Constructs to PHP Scripting Language

Control Structure Name	Description
require	Similar to include statement except that it generates a fata specified file cannot be found
include once	Similar to the include statement except if the file has beer evaluated by another file it will not be evaluated again
require once	The same as the require statement except, like the include requires the file to be read once
declare	This statement allows execution directives for a particular
foreach	Construct only designed to work on arrays. Allows a for l arrays but automatically calculated index ranges

A function in PHP looks very similar to a function in most programming languages. It is defined with the keyword function followed by the function name and a list of parameters enclosed within a set of parenthesis. The extent of a function is defined by a set of curly brackets. All the code for a function must be enclosed within these brackets. An added feature that is unique to PHP is the ability to write variable functions. A variable function is a function that is associated to a variable and can be called by simply appending a set of parenthesis to a variable. The PHP interpreter will then execute a function of the same name if it exists. Another unique feature not normally seen in most programming languages is the ability to write a function within a function. This would be useful when utility functions that are used multiple times but only within a particular function need to be created. Below in Figure 4, is an example of the syntax for a function.

```
<?php
Function DoSomething($arg1, $arg2, ….., $argn)
{
  // PHP variable declarations and code
}
?>
```

Figure 4: Function Declaration in PHP

The PHP programming language offers many advanced programming features like robust Object Oriented Programming (OOP) support, ability to throw and catch exceptions, and the use of references. More information on the discussed topics and advanced PHP features can be found in the online manual or in several books on the PHP programming language [Achour, et al. 2007][Sklar-Trachtenberg 2006 Sklar, David, et al., 2006].

2.2 Apache Web Server

The Apache Web Server is the most widely used web serving application on the web today and is freely available online [Apache Software Foundation 2007]. The first versions of the Apache web server were developed and released by Rob McCool at the National Center for Super Computing Applications (NCSA) which is a department at the University of Illinois, Urbana-Champaign. The initial release was sometime in 1994 and was simply called NCSA HTTP daemon. The history is not very well documented during it infancy. When Rob McCool left the NCSA towards the end of 1994 all development on the NCSA HTTP daemon stopped until a group of webmasters conversed via email about orchestrating some way to distribute the many additions and bug fixes that had been made independently. Over the next few months a core group of

webmasters added the bug fixes to NCSA HTTP v1.3 and released the first official version of the Apache Web Server in April of 1995. The name Apache was chosen for two reasons. The primary reason for the name was out of respect for the Apache Native American tribe and their resourcefulness and endurance through troubling times. Secondly, it was because it was created from a group of patches or in literal terms "a patchy server" [Apache Wikipedia 2007].

The initial version of the Apache Web Server was a huge success and became quite popular. To improve its scalability a major overhaul was performed to the server and resulted in Apache Web Server v0.8.8 which was released in August of 1995. This version sported a modular design and API, pool based memory allocation, and a new process model. Over the next few months, the development of standard modules was executed to add more functionality to the web server. At the conclusion of the module development and beta testing the first major version of Apache was released in December, 1995. Several more versions were released and as its popularity grew it became clear a more structured environment was needed leading to the formation of the Apache Software Foundation in 1999. The foundation helped focus the product development plus it provided both legal and financial backing to the effort. To better understand how PHP is integrated into the Apache Web Server we have to know the basics behind its architecture.

The Apache Web Server's architecture is modular in design. It allows for the creation of modules using a standardized set of function calls that are linked in at runtime. This makes it easy for new modules to be made by the Apache Software Foundation as necessary or a third-party in support of their product, which is how PHP is supported on all Apache Web Server ports. The callbacks provide a way to communicate with the core as shown in Figure 5. The Apache Web Server uses various functions or sometimes referred to as handlers [Dragoi 2006] within a module when they are required to service a client's request. A client request may consist of several phases that require multiple handler calls from several modules. For example, a PHP script within a file may be requested from a client. The first thing that might be done, based on the context of the request, is a handler function will get called to place the file in to memory. Next, the file would get passed to a handler function within the PHP module to be interpreted and once completed the handler function would return the data to the Apache core. The Apache core would in turn construct a message containing the return data to be sent to the client making the request. The client on the other end will receive the server message, the browser will read it and format it for display using local settings, and then display it to the client on their local machine. The basics behind Apache modules and how they are called have been described, but how does the Apache Web Server core receive client requests and decide what order to call the module handler functions?

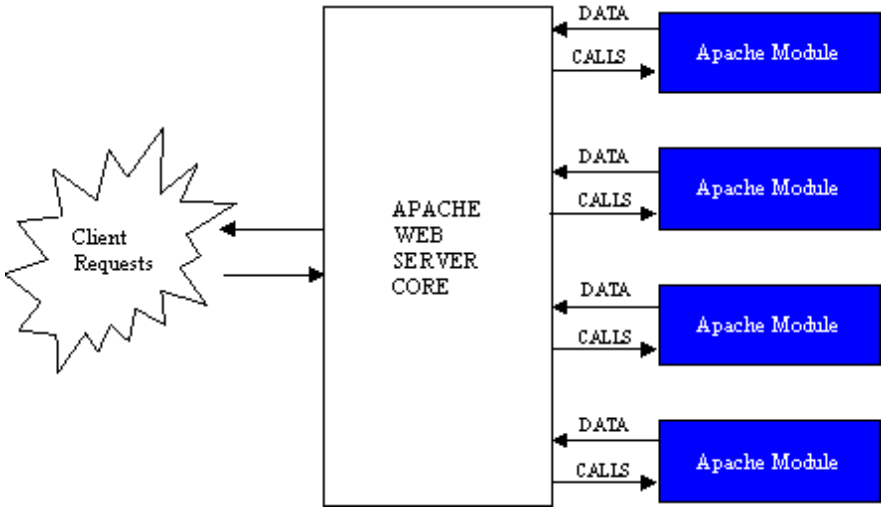

Figure 5: Modular Structure of Apache Web Server

The heart of the Apache Web Server consists of the core that processes all client requests. When someone opens a browser and types in a URL it gets converted to an IP address via a Domain Name Service (DNS).

The client browser then requests a connection to the web server address using port 80, which is the standard HTTP port number used to connect to all web servers. Once the connection is open the web server starts accepting requests from the clients for information from the website it is serving. When a request comes in there is a specified order in which the Apache core calls handlers within modules as shown in Figure 6. The modules contain private resources they use to complete operations and return data.

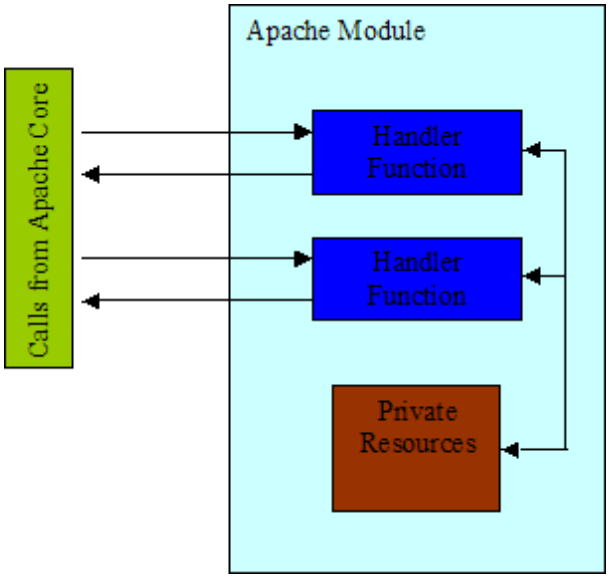

Figure 6: Handler Functions within a Module

Each handler does not know about other handlers and only takes information from and sends data back to the Apache core. The handlers that are called are based on the type of requests made by the client. In this paper, the main point of interest is when a client requests something that requires interpreting PHP code embedded in the website's source HTML. In some instances the PHP scripts will request data from the client. This data is received but not checked by the web server and then passed onto the PHP module that will use it to satisfy the original request made by the client. The next section will discuss how a mischievous client can use this ability to send unchecked data to exploit certain weaknesses within the PHP language.

---

3. Investigating PHP Vulnerabilities

Vulnerabilities in PHP can be in several different forms. The basic definition of vulnerability is some weakness in the system that allows someone to do something malicious to the system, which in this case is the web server. One form of vulnerability is via a poorly written PHP script by a user, which can be done by mistake or with malicious intent. Another form is by not understanding all the various settings that can be used with PHP and thus the administrator of the web server does not implement settings which are necessary for security. There are other vulnerabilities that exist which can cause a denial of service to the user (crashing the web server, flooding the network with traffic to where it is unusable, etc.). The following identifies some examples of these vulnerabilities and gives a more detailed explanation of each type of vulnerability.

3.1 PHP Script Vulnerabilities

While PHP scripts are not inherently bad, the user could either inadvertently write a script that allows the vulnerability or maliciously create a script to take advantage of certain PHP functions. A web administrator may not need to worry about such scripts for the main website that he/she manages, but other users on the

system which create and publish web content could create bad PHP scripts. The web server administrator must be aware of such issues with PHP so none of the pages on the system have vulnerabilities.

There are two major types of PHP vulnerabilities, local and remote. The local vulnerabilities are designed such that the attack is being made on the system on which the PHP script resides. A remote vulnerability executes a PHP script on one web server, which in turn goes out to get another PHP script and then is executed on the web server. In this instance, an attacker could place a PHP script on some other server which would cause vulnerability to a web server which executed it. Then the attacker can essentially tell the web server they are trying to exploit to grab that PHP file and execute it, causing the web server to be vulnerable. Specific local and remote vulnerabilities can be found in the following sections.

3.1.1 Local Vulnerabilities

There is really one major known locally vulnerable command in PHP. This command is the $_GET[] command, which allows the user to specify arbitrary values as variables through the URL [Achour, et al. 2007]. For example, if a HTML document had the following code shown in Figure 7 it would be considered a security risk [Wrenholt 2007].

```
<?php
$page = $_GET['page'];
if ($page)
    include $page;
?>
```

Figure 7: Example of File Inclusion Vulnerability

The $_GET command will take any variable in the URL called "page" and place it in the variable $page. Then $page is used with the include command. The include command will attempt to parse whatever is in the variable $page as a PHP script. If the file does not contain PHP script, it will basically just display the contents of the file. An attacker can then specify whatever they want in the URL and it will be shown on the web page. If an attacker specifies a URL like http://www.myhost.com/index.php?page=/etc/passwd and index.php contains the above code, the attacker will see the contents of "/etc/passwd" on the screen.

In order to still use $_GET without being vulnerable, the PHP scripter must sanitize the input to the command. Only specific inputs can be used with the statement and if anything else is specified the command will not be executed. An example of code that sanitizes the inputs for the PHP statement shown above is shown in Figure 8 [Wrenholt 2007].

```
<?php

$page = $_GET['page'];
switch ($page) {
	case "":
	case "home.php":
		include "home.php";
		break;
	case "about.php":
		include "about.php";
		break;
	default:
		echo "Invalid page";
		break;
}

?>
```

Figure 8: Properly Sanitizing Variables within PHP

this code only home.php or about.php can be specified. Every other input will be ignored (except if nothing is put in) then the default of home.php is used) so the script is no longer vulnerable.

3.1.2 Remote Vulnerabilities

Another way that a PHP script can introduce vulnerability is through the use of commands which get PHP scripts from other sites for execution locally. In this scenario, an attacker can set up their own web site which hosts a PHP script that contains a vulnerable command. The attacker can then tell the web site they wish to attack to get the vulnerable PHP script from the web site that they set up and execute it. One example of this type of attack is shown in Figure 9 [Clowes 2007].

```
<?php
 include($libdir . "/languages.php");
?>

<?php
 passthru("/bin/ls /etc");
?>
```

Figure 9: Example of Local Execution Vulnerability

The first PHP statement listed is on the server which the attacker wants to exploit. The variable libdir is defined on the server. An attacker could modify the libdir variable to put in whatever web site they would like to visit. In this case, if libdir was set to http://attackers.website.org and languages.php existed on that server, the server would go to http://attackers.website.org and attempt to execute languages.php on its own server. Within languages.php is the second PHP statement shown above. The passthru command locally executes

whatever is contained in the command as a shell script. In this case, the attacker would see the result of the shell command "/bin/ls /etc".

There are several different ways that this threat can be avoided. One way to alleviate this threat is to actually disable the passthru command. Another way to stop the threat is to sanitize the input to the variable in a similar fashion as was done for the local vulnerability (see Section 3.1.1).

3.2 Vulnerabilities Caused by PHP Configuration

Some vulnerabilities are introduced by incorrectly or not setting certain configuration parameters for the PHP module. Certain commands are not as well known as others or may not be documented properly. Many of these configuration parameters are known by the PHP community, but many web server administrators are not part of that community and do not know much about PHP. The configuration parameters could change and some could be added (or removed) at any time. The web server administrator would need to keep up with all of these changes. A few of the configuration settings which should be used are shown in Figure 10.

```
expose_php = off
php_admin_value file_uploads off
safe_mode = on
```

Figure 10: PHP Configuration Settings

To get rid of these vulnerabilities, the web server administrator would need to add, remove or modify the configuration settings defined in the configuration settings file (usually "php.ini") to reflect the correct values.

3.3 Other Vulnerabilities

PHP, along with most other applications, has problems that are inadvertently put in the source code which may cause the application to be vulnerable to certain attacks. Patches and bug fixes are released to fix the issues, but as development continues on the application, more problems can be introduced. The severity of the issue depends on the type of bug in the software. The vulnerability introduced could cause a Denial of Service (DoS) for the users, allow access to certain parts of the system which would compromise its security, give the attacker the ability to generate large amounts of network traffic to flood the network so it can no longer be used and so on. The best way to eliminate these bugs is to have the latest version of the software.

---

4. Vulnerability Detection Tool

Detection of vulnerabilities in PHP scripts and PHP daemon configurations are somewhat difficult and cumbersome tasks. For PHP scripts, a certain level of knowledge in PHP scripting is required to detect such vulnerabilities. Certain commands can be used safely in some contexts but could also be used maliciously or just by mistake by novice PHP scripters. On large web servers with hundreds of pages, the time to search through all the pages could be extensive. As for PHP daemon configurations, there are certain recommended settings and practices used by PHP experts to alleviate risks in PHP which the common system administrator may not know. Thus a tool has been created which recursively searches through HTML for certain PHP vulnerabilities and attempts to determine if the code is structured in such a way that it can be exploited. This tool also examines the PHP daemon configuration (and in some cases the HTTP daemon configuration) for recommended settings and notifies the user if these settings are incorrect or not present. The following subsections describe the usage of the tool and its design.

4.1 Basic Tool Design Considerations

Several issues were considered when designing the PHP VDT. Such issues included portability between operating systems, formatting differences in PHP scripts and PHP configuration files and flexibility for including new vulnerabilities as they become known. These issues are addressed in detail below.

4.1.1 Portability

One design consideration in the design of the tool was portability. PHP and HTTP daemons were created for many different operating systems. Thus the tool should ideally be made so that it can span all these operating systems as well. A portable coding language for use in creating the tool would be required to accomplish this. Therefore Java was used to code the tool so that it could be run on any platform which supported Java (which happens to be many of the platforms that support PHP daemons).

4.1.2 Syntactical Differences

PHP scripts and PHP configuration files can have different formats depending on the individual creating/modifying them. Certain PHP script commands and PHP configuration lines ignore white spaces. If this is the case, certain users may insert more spacing than others (or even use tabs) which could confuse a tool which attempts to parse the command or line. Thus the tool must be robust enough to be able to handle different formatting styles and command/line parsing must be independent of that style.

4.1.3 Flexibility to Include New Vulnerabilities

New vulnerabilities could become known at any time. Thus the tool must be able to handle the input of new vulnerabilities so that the user can take corrective action and not be susceptible to new threats. Currently this process is a manual process where the user must insert the new vulnerabilities into the tool by hand. A future extension of the tool could be to include a way to download a list of vulnerabilities which are automatically imported into the tool for use by the user.

4.2 Tool Requirements, Installation, and Start-Up

System requirements for running the tool are the same requirements which are needed to run Java. The user must install Java Runtime Environment, preferably the latest version. The tool has been tested with Java Runtime Environment (JRE) 6 Update 3. Once the JRE is installed, all the user needs to do is download the PHPVulnTool.zip file and expand it in the desired directory. Then run it using the following command:

java -jar PHPVulnTool.jar

4.3 VDT User Manual

Usage and configuration of the tool is fairly straightforward. Begin using the application by executing the JAR file with Java (see Section 4.2). The specification of the directory where the HTML files are stored and the file which contains the PHP configuration (or HTTP daemon configuration) is done before the vulnerability test can be run. Different settings can be specified which is broken down into two subsections: Search Rules and Configuration File Rules. The tool can then be closed by selecting File then Exit from the main screen. These pieces of the tool are explained in detail in the following subsections.

4.3.1 Specification of HTML Directory and Configuration File

In order to perform a recursive search on all the HTML files on the web server, the web server page storage directory must be specified. This can be done by either clicking on the "HTML Directory" button on the main screen (see Figure 11) or by manually typing in the directory in the field next to the "HTML Directory" button. When the tool is first opened, the "Progress" field will not contain any data. When using the "HTML Directory" button, a window will open that will allow the user to select the directory which contains the HTML files for the web server (see Figure 12). NOTE: User is only able to select directories in this window and not files.

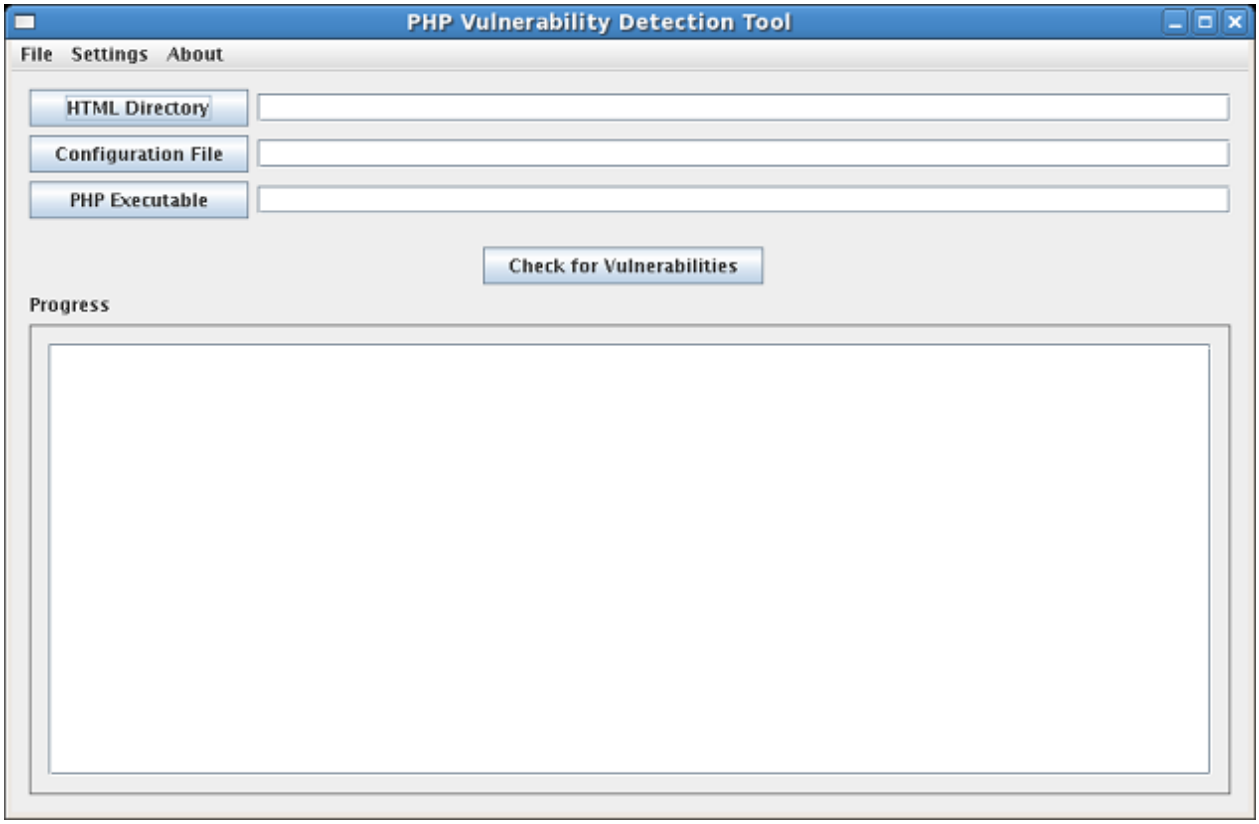

Figure 11: Screen of PHP Vulnerability Detection Tool

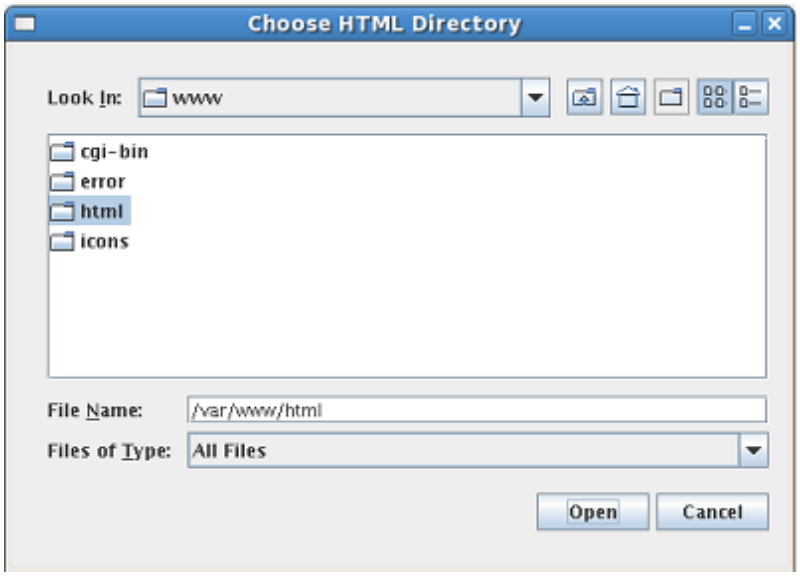

Figure 12: Choose HTML Directory Dialog

Once the user has selected an HTML Directory, a configuration file must be selected. The user may select the

configuration file for PHP (usually php.ini) by clicking on the "Configuration File" button and then selecting the file by browsing through the directories (see Figure 13) or by manually typing in the path and file name of the configuration file. NOTE:
Only a single file can be selected using this method multiple file selections and directory selections are not allowed. If another configuration file needs to be examined the tool must be run another time with the other configuration file selected.

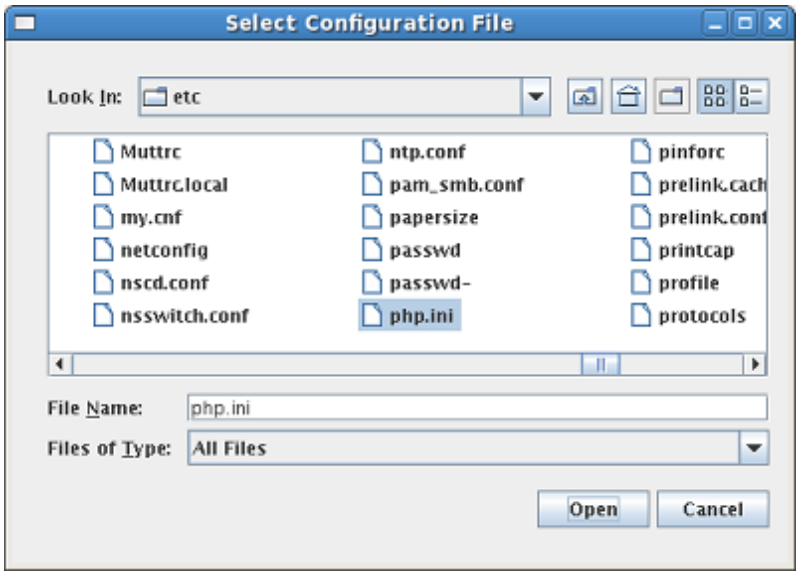

Figure 13: Select Configuration File Dialog

4.3.2 Search Rule Settings

One function provided by the tool is to search through each HTML file in a directory and each subdirectory within the directory selected (in the "HTML Directory" field, see Section 4.3.1). The rules which pertain to these searches are set in the Search Rule Settings box. This can be accessed via the menu selection Settings followed by Search Rules. The Search Rules Settings box is shown in Figure 14.

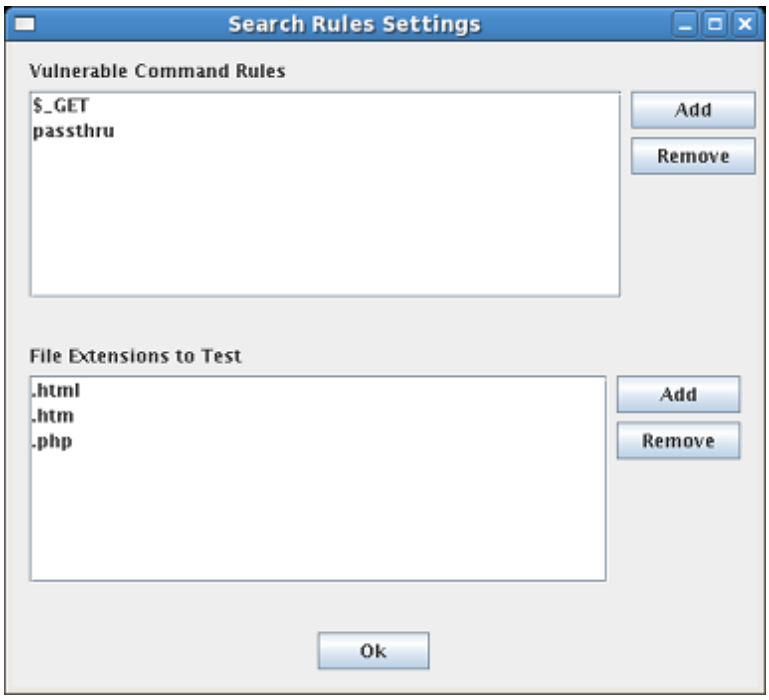

Figure 14: Search Rules Settings Dialog

This dialog contains settings for the vulnerable commands which can be found in the HTML files and also the HTML file extensions which should be tested for vulnerabilities. The default vulnerable command rules are "$_GET" and "passthru". These commands pose a potential security threat when used in certain ways. Thus the tool will search and detect these commands in the HTML files. Commands can be added to the list by selecting the "Add" button next to the command rules list. When "Add" is selected, a dialog box comes up which allows the user to enter in the command name for which to search. In order to remove commands from this list, simply click on the item (or multiple items) in the list and then click the "Remove" button next to the "Vulnerable Commands List" window.

The lower window of the dialog box shows the "File Extensions to Test" which are the file extensions that the vulnerability search will investigate for vulnerabilities. The default extensions (which are fairly common) are ".html", ".htm", and ".php". The user can add extensions to this list by clicking the "Add" button next to the "File Extensions to Test" window. File extensions can be deleted by selecting the appropriate (or multiple) file extensions that wish to be deleted and then clicking the "Remove" button next to the "File Extensions to Test" window.

4.3.3 Configuration Rule Settings

Another major function of the tool is to search configuration files for certain settings that would cause a PHP daemon to be more secure or more vulnerable. To view and/or change these rules, select Settings then Configuration File Rules... from the menu bar. The Configuration File Rules settings box is shown in Figure 15.

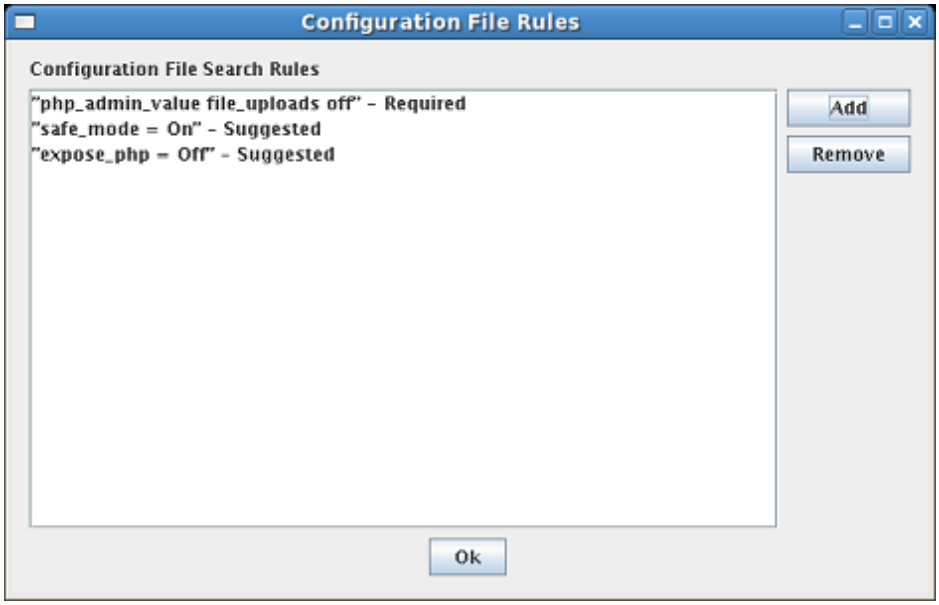

Figure 15: Configuration Rule Settings Dialog

Rules can be defined as required, suggested or disallowed. A rule that is required is a setting which is required to be present in the configuration file for security reasons and if it is not, a warning will be generated. Suggested rules are settings that are not absolutely required for security, but by enabling them it will moderately increase security. If a rule is disallowed, the setting indicated should not be in the configuration file and a warning will be generated by the tool if it is found in the configuration file.

Items may be added or removed from this list as required. To add a rule to the list, click on the "Add" button.

A dialog will pop up which asks for the rule to search for and then whether it is considered a required, suggested or disallowed rule. To remove a rule (or rules) from the list, select the rule (or rules) to remove and click on the "Remove" button.

4.3.4 Searching for Vulnerabilities

Once all of the settings have been made and the HTML directory and configuration file have been set, a scan for vulnerabilities can be completed. This is completed by clicking the "Check for Vulnerabilities" button in the main window (see Figure 4-1). A detailed report of progress is given in the "Progress" section of that same window. Progress indications are grouped into several categories: informational, advisories, and warnings. The following subsections will describe these message types, give examples of messages and describe what each message means.

4.3.4.1 Informational Messages

There are several informational messages which indicate progress of the tool to the user. Most of these messages just report information about what step the tool is currently at in the overall vulnerability check. A table of messages and their definitions is shown in Table 2.

Table 2: Informational Messages and Corresponding Definitions

Message	Description
Beginning recursive vulnerability check on "directory"...	Starting to recursively search the HTML directory "directory" for PHP vu
Checking in directory "directory"	Tool has entered directory "directory" and is beginning to search HTML directory for PHP vulnerabilities
Recursive vulnerability search on "directory" complete.	Completed the recursive search for PHP vulnerabilities on HTML files in "directory" and all its subdirectories
Beginning rules-based search on "file"...	Starting to search configuration file "file" for settings specified in the con rules
Rules-based search on configuration file "file" complete.	Finished search on configuration file "file" for rules specified in configura
Version of PHP at "file" is the latest version (<version>).	Returned version number matches the version expected
Vulnerability check complete.	Tool has completed all vulnerability checks

4.3.4.2 Advisory Messages

The tool will also inform the user when there is a possible vulnerability found that might require some action. These types of messages are called advisory messages. What action the user should take or if they should take action is sometimes dependent on what messages come after it. For instance, if a warning comes after an advisory message, the advisory message may give useful information about where to find the problem that corresponds to the warning message. A table of advisory messages and their descriptions is given in Table 3.

Table 3: Advisory Messages and Corresponding Definitions

Message	Description
Possible vulnerability found in "file", line number <number>	The HTML file "file" at line number <number> contains a possible vulner in Search Rules. Depending on how it is used, this may be a vulnerabili
Offending variable is <variable>	If the vulnerability assigns the result of a function call to a variable, the v <variable> is given. The tool will search the file to see if this is used in a causes vulnerability
Cannot read file "file"!	While attempting to search a file, the file "file" could not be opened
Error processing version number.	Unable to retrieve the version number from PHP executable
Version of PHP at "file" is version <old_version>, latest version specified as <new_version>! Upgrade to the latest version of PHP!	The current installed version is early then expected safe version numbe

4.3.4.3 Warning Messages

Messages which indicate a problem that the user needs to address are warning messages. These messages indicate that either vulnerability in a HTML file has been verified by the tool or that a required setting or disallowed setting has been found in the configuration file. The user should make changes to the appropriate files to correct these issues. The list of warning messages and their descriptions can be found in Table 4.

Table 4: Warning Messages and Corresponding Definitions

Message	Description
WARNING: Variable <variable> is vulnerable!	A possible vulnerability in variable <variable> that was found by the tool being vulnerable because of its usage in the HTML file.
WARNING: Required rule <rule> was not found!	Configuration file rule <rule> that was identified as required was not fou configuration file.
WARNING: Disallowed rule <rule> was found at line <line>!	A disallowed rule <rule> was found in the configuration file at line numb

5. Summary

The combination of the Apache Web Server and PHP scripting language are powerful tools when creating websites for all types of purposes. Within the PHP scripting language there are powerful constructs for interfacing with databases, complex file parsing, talking with other services like email and COM, and even tools to create GUI applications. Just as in real life, with great power comes great responsibility and if web developers are not careful they can leave their web servers vulnerable to several different types of attacks by malicious clients. These attacks can range from minor annoyances to bringing the whole system to a standstill. The worst types of script vulnerabilities are the ones that allow a malicious client to gain access to the web server by obtaining password hashes or allowing malicious code to be downloaded and executed on the web server. If the malicious client gains access to enough resources on someone else's web server they can even launch attacks on other websites. This paper has attempted to give the reader a general knowledge of the PHP scripting language and the basics behind a popular web servers operation. Using this knowledge, an investigation of several PHP vulnerabilities was conducted and information was given that web developers can use to avoid writing PHP scripts that can be exploited. In addition, a simple Java based utility was created by the authors that checks for basic vulnerabilities in files that constitute a website. These files by default end with the PHP or HTML extension but other types can be added as well. Likewise, there are some default rules

that are checked and when violated flagged with a predefined severity. Additional rules and associated severities can be added by the user for other vulnerabilities in PHP, HTML, or other types of scripting languages as they are discovered. The severities range from suggestions on setup settings to disallowing the use of certain types of statements in scripts. Future improvements on the tool could include more robust rule checking logic since the initial version is quite limited on its ability to catch certain types vulnerabilities that may occur via several layers of code.

---

6. References

[Achour et al. 2007] Achour, Mehdi, Betz, Friedhelm, Dovgal, Dovgal, Antony, et al., "PHP Manual", Oct. 2007, http://us.php.net/manual/en/index.php/

[Apache Software Foundation 2007] The Apache Software Foundation, "Apache HTTP Server Project Website", Copyright 2007, http://httpd.apache.org/

[Oglio et al. 2006] Oglio, Pablo Dall, Mattocks, Scott, Narayanan, Anant et al. "PHP-GTK Manual", Copyright 2001-2006, http://gtk.php.net/manual/en/

[Sklar-Trachtenberg 2006] Sklar, David, Trachtenberg, Adam, "PHP Cookbook 2nd Edition", O'Reilly 2006

[James-Ware 2002] Lee, James, and Ware, Brent, "Open Source Web Development with LAMP: Using Linux, Apache, MySQL, Perl, and PHP", Addison Wesley Professional 2002

[Dragoi 2006] Dragoi, Octavian Andrei, "The Conceptual Architecture of the Apache Web Server", Department of Computer Science, University of Waterloo, 2006, http://www.grad.math.uwaterloo.ca/~oadragoi/CS746G/a1/apache_conceptual_arch.html

[Apache Wikipedia 2007] Anonymous, Wikipedia on Apache Web Server, http://en.wikipedia.org/wiki/Apache_HTTP_Server

[Wrenholt 2007] Wrenholt, Erik, "Writing Secure PHP Applications", 2007, http://www.timestretch.com/site/writing_secure_php_apps/

[Clowes 2007] Clowes, Shawn "Exploiting Common Vulnerabilities in PHP Applications", 2007, http://www.secureреality.com.au/studyinscarlet.txt

---

Download VDT

Select the link below to download the zip file containing the Vulnerability Detection Tool v0.1
Vulnerability Distribution Tool v0.1 (101KB)

List of Acronyms

DNS - Domain Name Service
DoS - Denial of Service
HTML - HyperText Markup Language
HTTP - HyperText Transfer Protocol
JAR - Java ARchive
JRE - Java Runtime Environment
NCSA - National Center for Supercomputing Applications
OOP - Object Oriented Programming
PHP - PHP Hypertext Preprocessor
PHP/FI - Personal Home Page / Forms Interpreter
PHP - GTK - PHP Hypertext Preprocessor Graphical Tool Kit
STL - Standard Template Library
VDT - Vulnerability Detection Tool

Back to Table of Contents

Last Modified: December 2, 2007